Lessons Learned in Building a Middleware for Smart Grids

Marcel Macarulla[1], Michele Albano[2], Luis Lino Ferreira[2] and César Teixeira[2]

[1]*GRIC, Department of Project and Construction Engineering, Universitat Politècnica de Catalunya, Spain*
[2]*CISTER, ISEP/INESC-TEC, Polytechnic Institute of Porto, Portugal*
E-mail: marcel.macarulla@upc.edu; {mialb; llf; crdst}@isep.ipp.pt

Received 22 April 2016; Accepted 30 May 2016;
Publication 4 August 2016

Abstract

Smart grids play an important role in the modernization and optimization of the existing electrical grid, to accomplish the current European Union Energy and Climate targets. Smart grids require distributed applications to manage the grid more efficiently. The performance of the distributed applications impacts on the communications delay time and on the timely interaction with the devices located in the users' Home Area Networks. This paper presents the results of the ENCOURAGE project related to the development of a software platform to support smart grids. The work presented in this paper assesses four different middleware configurations and analyses the results on the delay performance tests. The results show that the mean end-to-end delay is between 310 ms and 453 ms in proper conditions. In terms of operational costs, the optimal configuration enables managing houses with less than 0.25 Euros per month per house. This paper justifies the maturity of the technology to support smart grids, and the possibility to transfer the ENCOURAGE project results to the industry.

Keywords: Smart grid, middleware, publish/subscribe, performance.

Journal of Green Engineering, Vol. 6, 1–26.
doi: 10.13052/jge1904-4720.611

1 Introduction

The evolution in computing technology is shaping the society we live in. Distributed applications are realized for both computational power (e.g.: cloud computing) and to embed data collection devices within the physical environment (e.g.: wireless sensor networks). In both cases, distributed applications offer robustness and performance at the cost of the complicacies to build and optimize them. Mediating all the interactions via a middleware transporting a single protocol can scale down the system complexity, but this moves complexity from the technological plane to the political one, since a long tradition of standardization efforts shows how hard it is to agree on one middleware and one data representation for an application area as complex as the smart grid [1].

The performance of the different configurations that can be used in a distributed smart grid was seldom measured, and this paper has the intent of comparing different designs and middleware configuration supported on real scenarios. Previous work [2] has identified in messaging-oriented buses a valid family of middleware for smart grids. In fact, they enable a good level of decoupling of the identity of data producer and consumer, performance, and support Quality of Service in terms of prioritization of the information flows.

Some studies have analyzed the performance of messaging systems within toy deployments [3, 4]. Since most modern middleware use XML-based encoding for data, some works have studied the time complexity of marshalling/unmarshalling operation, which is the translation of in-memory data structures into XML strings, and vice versa [5]. This paper is focused into performing an analysis of a real smart grid, by means of tests on the code[1] of the middleware that was developed – and deployed – over a real scenario.

This paper analyses the results of the European project ENCOURAGE [6], and the lessons learned in the process, while building a software platform to support a smart grid. To this purpose, after providing background on smart grids and related middleware (Section 2), we model a typical smart environment (Section 3) and we discuss different design choices that can impact the performance of the main software modules involved in the distributed system (Section 4). Then we submit the system to an extended set of tests, with the objective of identifying bottlenecks caused by communication and computational costs (Section 5). We finally draw conclusions on which

[1]The code was recently made opensource, it is available at:
https://github.com/cistergit/ENCOURAGE

approaches are better and in which scenarios (Section 6), and in particular we show that the most common approaches, recommended for example by RabbitMQ site [7], lead to low performance over real scenarios.

2 Background Information

2.1 Smart Grids

The European Union has ambitious Energy and Climate targets for 2020 and beyond to reduce greenhouse gas emissions, increase the share of renewable energies and improve energy efficiency, and it is reflected into the energy policy of European countries [8]. In this context, the concept of smart grids plays an important role in the modernization and optimization of the existing grid. In the conventional grid the energy flow was unidirectional and the communication signals were used only for billing purposes. The main contribution of smart grids to conventional power grids is to provide bidirectional flow of energy and communication signals [9], and limit periods of strain on network and electricity markets [10].

The bidirectional flow of energy enables the integration of distributed generation into the grid, bringing the opportunity to introduce local energy production based on renewable resources in the system, and reducing the carbon emissions [11]. Storage systems can also be introduced into the grid thanks to the bidirectional flow of energy. The use of storage systems can help to integrate renewable energy resources, storing the energy that cannot be sold because of the limits on transport capacity (i.e. an energy surplus) or because there is a lack of energy demand [12].

In terms of communications the bidirectional flow enables different new functionalities and services that can potentially be offered to the consumer. Examples of these new functionalities due to the bidirectional flow can be, the temporal limitation of the maximum electricity consumption, the termination or re-connection electricity supply to any customer remotely [13] and the identification and monitoring of individual appliances in a household using nonintrusive load monitoring [14].

At European level different national and regional smart grids initiatives exists [15]. The majority of western European countries are already transitioning form Research and Development (R&D) projects to Demo and Deployment projects (D&D), demonstrating the maturity of the smart grid technologies and applications. Denmark and Germany are the European leaders in R&D and D&D projects [15]. One of the biggest projects is

Smartcity Málaga, with 11 companies enrolled, covering 4 km^2, involving 11.000 domestic customers, and 1.200 industrial and service customers. The Smartcity Málaga project [16] covers technologies such as smart metering, communications and systems, network automation, generation and storage, and smart recharging infrastructure for e-vehicles.

2.2 Advanced Message Queuing Protocol (AMQP)

Since a smart grid is a complex system where many actors interact, on both the data plane and the energy plane, data must be managed in a performant manner that allows decoupling in terms of space, time and synchronization. Space decoupling means that publisher and subscriber do not need to be aware of each other's location or identities. Time decoupling means that publisher and subscriber do not need to be online and actively collaborating in the interaction at the same time. Synchronization decoupling allows asynchronous notification of subscribers, for example by using event services callbacks. A message-oriented messaging bus allows the decoupling needed by this scenario [17], and the Advanced Message Queuing Protocol (AMQP) is a good candidate for vehiculating smart grid data and commands.

The Advanced Message Queueing Protocol (AMQP) is an open standard application layer that allows message exchange between applications or organizations. A number of tutorials [7], research papers [18] and whitepapers [19] have provided in-depth descriptions of AMQP. This section briefly describes its main concepts, and we leave the reader to the cited material to gain further insights on the protocol.

In the AMQP protocol, interacting parties consist of brokers, producers, consumers, exchanges, queues and bindings. Producers and consumers are respectively applications that publish messages into the broker, and receive messages from the broker. Message brokers act as bridges between two applications, receiving messages from producers and routing those received messages to the consumers. The publisher sends messages to an exchange in the broker. The exchange is responsible for routing copies of the messages to queues in the broker. Consumers receive messages from the queues. The routing of messages from the exchanges to queues follows some pre-defined rules, called bindings, which define the connection of a queue with a specific exchange. Figure 1 presents a basic idea of how the AMQP protocol works as bridging applications.

Consumers and producers communicate with the AMQP broker by means of AMQP channels, which transport the messages. Channel are hosted into

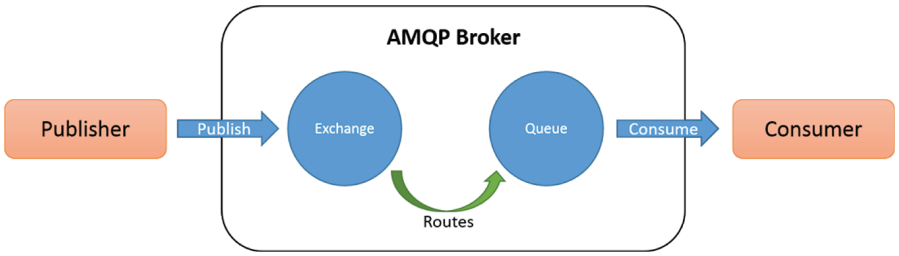

Figure 1 Advanced message queueing protocol architecture.

AMQP connections, which are basically TCP/IP connections. By hosting different AMQP channels used by the same application into the same AMQP connection, network resources of the operating system can be saved [3].

Messages, exchanges and queues can be associated with meta-data, which define different characteristics of the communication. For the goals of this work, the most interesting meta-data regard message persistency, message acknowledgment, queue prefetch count, and virtual host.

If the message is persistent, the message is cached on local persistent storage until it is delivered to the consumers, thus the message will be recovered and delivered even if the message broker fails in the meantime.

To ensure robustness of communication, AMQP provides message acknowledgement techniques. The broker repeats sending the message to the consumer until it receives an acknowledgment message from the consumer itself.

The prefetch count attribute specifies how many messages are consumed at a time. Each consumer maintains a local buffer containing a number of messages equal to the prefetch count attribute, ensuring that the consumer has got computational work to do even when communication problems do not allow for fast reception of new messages. A proper value for the prefetch count can reduce the total time needed to complete the distributed computation, and the response time of the distributed system.

Each AMQP broker can serve multiple application domains. To enable the separation of subsystems, the virtual host attribute allows the creation of multiple virtual AMQP brokers that embrace different groups of users, different exchanges and different queues.

A number of brokers implement the AMQP protocol. The work described in this paper makes use of the RabbitMQ implementation [7], which appeared to be the most mature and stable at the time of the design of the system.

2.3 The ENCOURAGE Project

The ENCOURAGE – acronym for Embedded iNtelligent COntrols for bUildings with Renewable generAtion and storage – Project [6] developed a platform that contributes for the optimization of energy usage in buildings and participate in smart grid environments. The goals of the platform are a 20% saving on energy consumption, in line with European targets [8], and the ability to have a systematic and a performant monitoring system that provides near real-time information regarding all devices, presenting data to users through web-based interfaces, reports, and social network technologies.

The strategies to optimize energy usage belong to three families. In order to save energy consumed by large subsystems, such as HVAC (heating, ventilating, and air conditioning systems), lightning, renewable energy generation and thermal storage, a set of supervisory control strategies were developed. Since most nefarious effects of energy consumption are related to peak consumption [20], the same supervisory control strategies are applied to flatten out peak demands by orchestrating the large number of appliances that are consuming energy in the time period, while taking into account user's comfort, and the energy costs, and leveraging on peoples' habits, weather forecasts, characteristics of appliances, local generations and storage of energy, and market conditions. Finally, a virtual representation of every device within the platform and the design and implementation of an event-based middleware enables real-time advanced monitoring and diagnostics capabilities of the smart grid, and allows the usage of business intelligence mechanisms [21].

3 System Model

Let us consider a smart grid topology where sensors/actuators are deployed in the final user's house and communicate with a gateway, which connects to the internet. This topology assumes the denomination of Home Area Network (HAN). The functional requirements that need to be satisfied by a general purpose smart grid belong to three families:

- Communication: Data are collected from the HANs of the users. Commands and energy prices are transmitted to the HANs of the users, to impact on the usage of appliances.
- Online Control: A smart grid component computes in real-time the strategies to be implemented to improve energy usage, based on the data collected in the HANs.

- Offline Analysis: Data regarding events happening in the smart grid are stored, to be periodically analysed and presented to the users.

The smart grid architecture analysed in this paper is focused on satisfying the three requirement families described above. Each HAN gateway is integrated with a **Middleware Plugin** (MPG) that translates the custom protocol of the HAN into a common protocol, and connects to a cloud. The cloud hosts a messaging system that binds together the "smart" parts of the smart grid. The cloud hosts a **Virtual Devices** (VD) module that routes messages and caches them, a **Supervisory Control** (SC) module that schedules energy usage and drives users' appliances, and a **Database Handler** (DBH) that enables the permanent storage of data for further processing and visualization. The MPG, the messaging bus and the VD module take care of the Communication requirements, the SC is devoted to the Online Control, and the DBH enables the Offline Analysis.

For maintainability and compatibility with current and future systems, the smart grid translates all data collected in the HANs and all the commands for devices in the HANs into an international standard, the IEC Common Information Model (CIM) [22], which uses XML to encode the data. The MPG encodes all data into CIM (marshalled) for transmission over the distributed system, and then unmarshals them back to the corresponding in-memory data structures, with both the operations supported by the JAXB [23] library. With respect to the Common Information Model, exchanged data are either MeterReadings (sensor data collected in a HAN, or weather information) or EndDeviceControls (commands sent by the SC Module), both belonging to the IEC 61968 standard, which is part of the CIM.

The components are connected via a publish/subscribe messaging bus built upon the RabbitMQ [24] implementation of the Advanced Messaging Queue Protocol (AMQP) [19] protocol. The structure of the exchanges and queues is presented in Figure 2, which features the modules described above (MPG, VD, SC and DBH).

The VD caches in-memory the current state of the HANs and acts as routers between SCs, MPGs and DBH. The VD is instrumental for the smart grid, since current home automation and smart building infrastructures are commonly developed and implemented as separate systems with their own user interfaces and gateways (e.g. HVAC systems, water/gas/energy metering systems, etc.). The VD module allows to have multiple gateways in each HAN, each responsible for a subset of the HAN devices, and provides a logical vision centred over the HANs. VD takes care of dividing collected data into logical

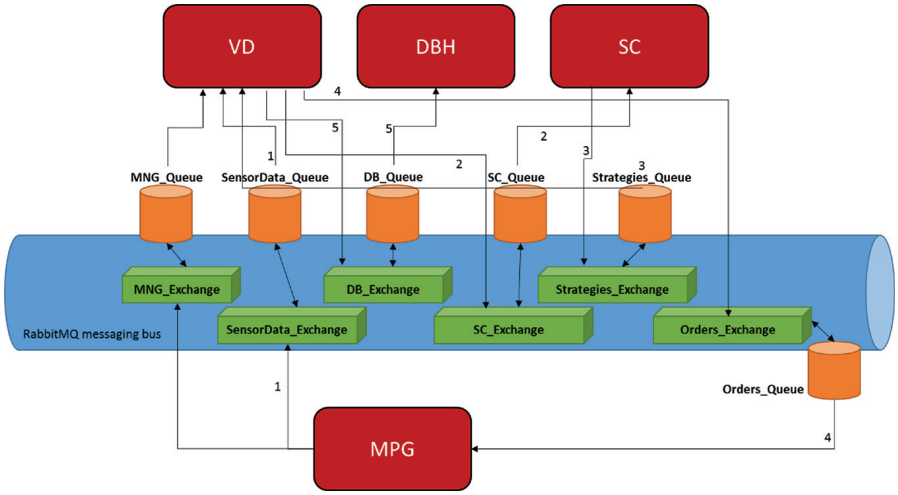

Figure 2 Routing structure of the ENCOURAGE middleware.

sets pertaining to the HAN, and reach back the correct HAN gateway when a command is available.

The SC receives HAN data from the VD, computes an energy-saving strategy, and sends commands back to the VD. Since the VD keeps track of which SC is serving a given HAN, it is possible to have multiple SCs active in the smart grid at the same time, to speed up the system by means of parallelizing the computation of energy-saving strategies.

The DBH module receives data regarding each event in the smart grid (sensed data, energy strategies, weather forecasts) and interacts with a database to implement the permanent storage of the data.

With reference to Figure 2 when data is collected in a HAN, the most complex chain of events is as follows. Data collected from the HANs are encoded into CIM and published to SensorData_Exchange by a MPG (flow 1), and routed by the VD module to SC_Exchange (flow 2). The message is then received by the SC module, which computes energy-saving strategies and publish them onto Strategies_Exchange (flow 3). The message is then routed by the VD module to Orders_Exchange (flow 4) with the correct routing parameters, and received by one of the MPGs, which implements the control on the devices in the HAN. Moreover, VD Module consumes from all the exchanges of the messaging server and it pushes messages to the DB_Exchange to reach the DBH (flow 5), which stores event data in the database. In the following, this chain of event will be called “full smart grid workflow”.

Apart from the previous workflow, which represents data communication in the smart grid, the messaging broker hosts the MNG_Exchange, which is used by applications to modify the configuration of the smart grid and its modules, for example in terms of topology.

Next section introduces the performance challenges that the ENCOURAGE smart grid coped with, it proposes a number of different designs for the modules, and it discusses them and the parameter space that was investigated while building the module.

4 Internal Structure of the Modules

This section presents an analysis of the critical paths and potential bottlenecks in the ICT platform serving the smart grid, and the strategies to improve its computational performance. The goal of the analysis is to maximize the message throughput of the smart grid, and to minimize the end-to-end delay, both with regard to the full smart grid workflow (see end of Section 3).

In principle, a larger delay can lead to a lower message throughput, and the two issues are discussed together in the following. There are two sources of delay in a distributed system: data processing and data transmission.

Data processing delay is related to the computation of the energy consumption strategies, to the marshalling/unmarshalling of the messages, and to the management of the concurrent actions being taken by the modules (thread structure, etc). This work focuses on the middleware transporting data in the smart grid, thus the SC is out of the scope of the discussion, and the SC is simulated by assigning it a fixed delay (100 ms). Marshalling/unmarshalling are CPU-intensive operations and central to the functioning of the modules. The operations are based on the JAXB [23] library, which is based on DOM [25] and thus possesses a great deal of flexibility for message processing but can potentially present performance issues. Finally, the thread structure of the modules can lead to overheads that impact the computational performance of the system.

The transmission delay depends on the performance of the network connecting the system components to the AMQP messaging bus, and how the messaging bus itself is used. The messaging bus in use (AMQP's implementation RabbitMQ) has got features that can affect both the system performance and communication robustness (see Subsection 2.2).

Other operations can be performed by the system, such as computation of reports for the final user, storage of consumption data on DBs, visualization

of the data. Still, they are either lightweight, or are performed at the end of the full smart grid workflow and thus do not represent performance bottlenecks.

All modules of the smart grid make use of a specialized library, called Encourager library, that abstracts the applications from every aspects related to marshalling/unmarshalling the messages, management of the internal structure of the programs (thread structure, etc), and configuring and interacting with the messaging system, such as setting parameters on the RabbitMQ exchanges and queues. The library allows the modules to be built without getting concerned with low-level details common to all modules, and to apply strategies to improve system performance to all modules by modifying the library's code only.

Regarding the transmission delay, the parameter space considered in the analysis consists of the RabbitMQ options on message persistency, message acknowledgment, queue prefetch count, and virtual host of the RabbitMQ implementation of AMQP protocol (see Subsection 2.2). Regarding data processing delay, the following discusses different designs that were implemented in the Encourager library, and indirectly in all the smart grid modules.

4.1 Dynamic Design

Figure 3 depicts the first design studied for the Encourager library, named the "Dynamic Design". In this approach, one thread is created for each element that interacts with the messaging system, to maintain its state. In the case of the VD module, it means that each sensor and actuator in the smart grid is

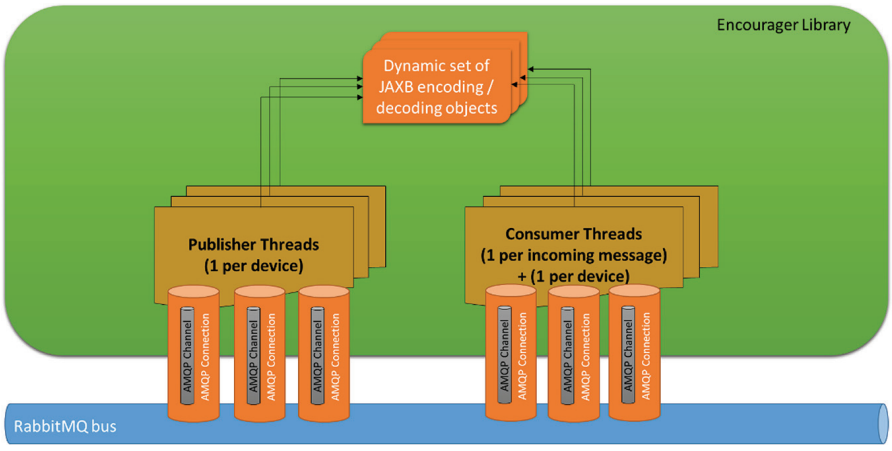

Figure 3 Encourager library, dynamic design.

associated with a thread. Furthermore, each time a message is published or received, a new thread is created to manage the message, and a whole set of JAXB marshalling/unmarshalling objects is instantiated.

The benefit of this approach is that the computations related to messages are isolated from each other, in agreement to most common modern programming paradigms [26]. On the other hand, if many messages are received in a short time, the large number of threads that are created can lead to malfunctions, and the time used for processing each message can be quite large since each message causes the initialization of a new set of JAXB marshalling/unmarshalling objects. Note that this strategy for message management is the one suggested by the RabbitMQ documentation [7].

4.2 Massively Multi-threaded Design

This alternative structure for the Encourager library, named "Massively Multi-threaded Design" and represented in Figure 4, considers having the same thread structure for the elements communicating with the messaging bus, but sharing a single JAXB marshalling/unmarshalling infrastructure for the decoding of all the messages. Since the JAXB infrastructure is thread safe, this approach did not impair any functional or non-functional requirement. Moreover, no thread is created for each incoming message, and instead the message is processed by a thread representing a device.

In this approach, threads that are not currently processing data can block to save system resources. Anyway, in the particular case of the VD module, the

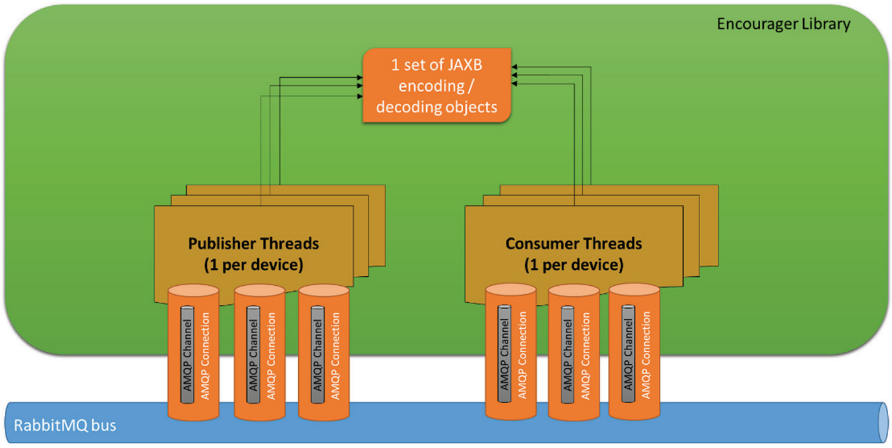

Figure 4 Encourager library, massively multi-threaded design.

number of threads grows as the number of sensors and actuators in the smart grid. Even though in most traffic conditions a small number of threads are active at the same time, heavy traffic could lead to high resource consumption and potentially to system malfunctions.

4.3 Multiple Channels Design

The approach "Multiple Channels Design", represented in Figure 5, considers having a fixed number of threads in the system, contained in a thread pool, to have a stricter control on resource utilization. Two AMQP connections (see Subsection 2.2) transport all incoming and outgoing messages respectively. Each Consumer Thread is associated with an AMQP Channel, thus the set of Consumer Threads ends up using a constant set of AMQP Channels. On the other hand, as suggested by the RabbitMQ documentation [7], a new AMQP channel is dynamically created every time a message is published.

Data related to entities communicating with the RabbitMQ messaging bus, for example representations of sensors and actuators in the VD module, correspond to data structures in memory. One thread receives all incoming messages, and dispatch them to worker threads that take care of decoding and processing the message. The worker thread takes control of the data structure of an entity, updates the memory representation of the device and performs proper actions over other modules of the distributed system, for example by sending messages to SC, MPG or DBH.

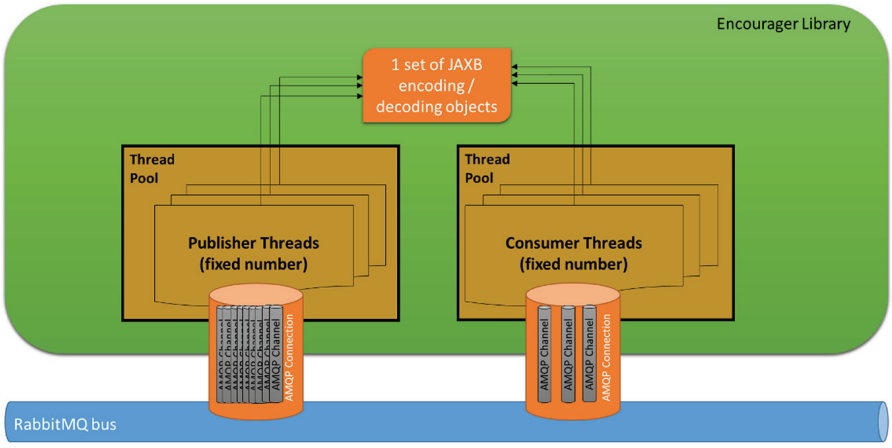

Figure 5 Encourager library, multiple channels design.

The limited number of threads used in this approach could lead to issues in exploiting the multiprocessor/multicore capabilities of some computational platforms, still the resource usage cannot get out of control like in the previous designs.

4.4 Application-layer Dispatching Design

This last strategy, named "Application-layer Dispatching Design" and depicted in Figure 6, is a minor improvement over the Multiple Channel Design. The only difference with the previous approach is that every outbound message is published through the same channel, with the aim of saving over the time needed to set up a new AMQP channel.

5 Evaluation Results

This section discusses the results of the evaluations that were performed on the system. The parameter space that was investigated comprises the network options and architectural decisions described in Section 4, and the dimensioning of the system. The network parameters regarded how the Encourager library interacts with the AMQP broker (see Subsection 2.2), in particular the options on message persistency, message acknowledgment, queue prefetch count, and virtual host. The architectural decisions under exploration were described in Section 4. The dimensioning of the

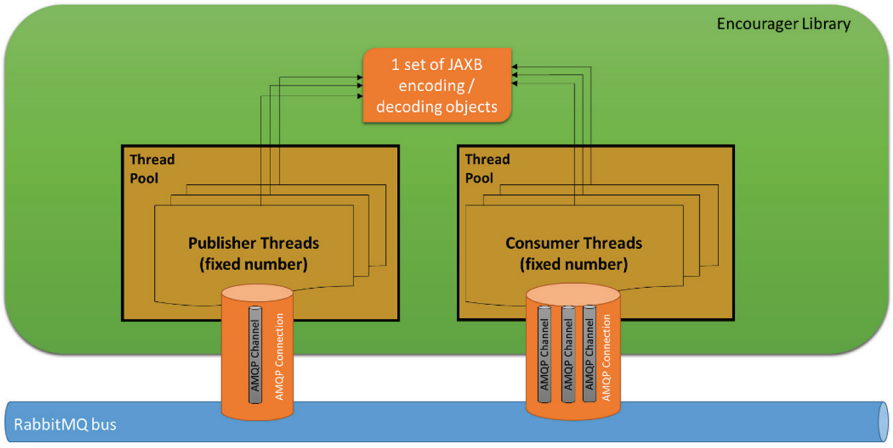

Figure 6 Encourager library, application-layer dispatching design.

system considered the performance of the smart grid against the number and locations of MPGs and simulated sensors and actuators in the system, the rate of the messages, and the size of the messages containing sensed data.

The experiments measured the time passed between the publication of a sensed data by the MPG, and the reception by the MPG of a command generated as an answer to the sensed data. With reference to Section 3, the data at hand is the time elapsed for the smart grid to perform a full smart grid workflow.

The section starts with the results of a preliminary analysis, which laid the foundations for the rest of the discussion. Later on, we put the smart grid under saturation condition to identify bottlenecks and limits on the performance. Finally, we considered scenarios where the smart grid was close to the identified limits, both to verify them and study the performance of the smart grid in a realistic scenario.

5.1 Preliminary Analysis

The preliminary analysis aimed at lowering the complexity of the parameter space under study. We considered the parameter space under analysis, and we either ended up with reasonable fixed values for some parameters, or decided which value ranges were going to be under scrutiny.

In every test, we considered a full smart grid workflow (see end of Section 3), which comprises the reception of data sensed in the HAN and the delivery of load control commands to the HAN's actuators. A smart grid that has a proper performance in response to a full smart grid workflow has got sufficient performance for a real scenario, since the full smart grid workflow is the most complex chain of events that can happen for a HAN. On the other hand, the support of full smart grid workflows for all the MPGs of the smart grid in a time period is also necessary, since integration of renewable energy sources [27] can involve the remote control of all the MPGs of the smart grid at the same time.

Preliminary experiments showed that the computing node hosting the VD module was the one under the heaviest computational stress. This hints that the VD module can be considered the computational bottleneck of the smart grid, and this is due to the computational load of the operations it performs (unmarshalling and marshalling of every message exchanged in the smart grid, decisions regarding the recipients of further messages). Thus, the performance experiments aimed at measuring the size of the smart grid that

could be supported by one VD module executed on a computing node of a fixed computational power. We decided for setting up a virtual machine executing just the VD module, and having the same configuration of the lower-end (t2.medium) virtual machines rented at Amazon [28].

Regarding the dimensioning of the rest of system, the system featured one SC, simulated by means of a fixed delay of 100 ms, and one DBH, intending that they can use the elasticity of a cloud when under heavy stress. Preliminary experiments using 1, 3 and 10 MPGs (installed on different computers) and 1, 30, 242 sensors/actuators showed that the performance was impacted by the rate of the full smart grid workflow only, and its behaviour was irrespective of the number of MPGs and sensors/actuators. Thus, in the rest of the paper, we considered experiments with one MPG, and 242 simulated sensors and actuators.

To measure the performance of a geographically distributed smart grid, we deployed the modules of the smart grid on different locations in the Iberian peninsula. The VD module was deployed on a virtual machine in the Polytechnic Institute of Porto, Portugal. The SC module, the DBH module and the RabbitMQ messaging bus were deployed in the campus of the Catalunya Polytechnic, Terrassa, Spain. The MPGs were deployed either on a computer located on the Polytechnic Institute of Porto, Portugal, or in houses in Portugal connected to the internet using ADSL technology.

In the rest of the paper, we will consider that the message containing data sensed in the HAN can be either big or small, containing 50 or 1 measurements respectively. The control message generated by the SC contained always one demand response command.

Regarding the RabbitMQ options, the preliminary experiments proved that the usage of a virtual host added flexibility to the system without any perceived overhead. The message acknowledgment proved to be necessary to add proper robustness to the system communication. A prefetch count equal to 20 was able to maintain a high level of processor utilization on the system hosting the VD Module. Message persistency was not important in our scenario, and it introduced heavy performance overhead, thus we opted for using non-persistent messages.

Next sections will thus consider the big/small measurement messages described above, the four alternative designs for the Encourager library described in Section 4, one MPG located either in the campus of Polytechnic of Porto, Portugal, or behind an ADSL in a house in Portugal, and different rates for the full smart grid workflows.

5.2 Performance Under Saturation

These experiments considered to put the system in saturation by providing a large number (10000) of messages containing HAN sensor data, all generated by the MPG module at the same time, to measure the bandwidth of the whole system. The system being in saturation, each message requires a longer time to be processed than the previous message, since the new message was going to be processed after all the previous ones. The hypothesis that was tested is that it is possible to compute the bottleneck time (time delay introduced by the slowest processing step) by dividing the total time needed by the system to complete its work by the total number of full smart grid workflows.

The Dynamic Design (Subsection 4.1) highlighted a problem on the operating system level. Since each time that a message was received by the VD module one thread was created, the VD module created too many threads and crashed before completing the handling of 6000 small messages. The Dynamic Design proved to be a hurdle for system stability, and it will not be considered anymore in the paper, also because it was not possible to collect data for the 10000 messages exchanged in each experiment.

Table 1 reports the mean delay introduced when starting the full smart grid workflow with small measurement messages, with the VD module deployed on a university campus or behind an ADSL (see Subsection 5.1), and Table 2 reports the same kind of data in the case of big measurement messages. Each experiment was repeated 20 times, and mean values are reported. All of the analysed approaches did not impair any functional requirement of the system. On the other hand, the results corroborate the idea that investment into the correct programming of the underlying computational support can provide a strong speed-up to the data layer of the smart grid. Moreover, the results show

Table 1 Full smart grid workflow processing time in case of *small* measurements, in milliseconds

	Campus [ms]	Home ADSL [ms]
Massively Multi-threaded Design	190.3	221.0
Multiple Channels Design	149.6	158.1
Application-layer Dispatching Design	7.5	7.7

Table 2 Full smart grid workflow processing time in case of *big* measurements, in milliseconds

	Campus [ms]	Home ADSL [ms]
Massively Multi-threaded Design	229.3	235.5
Multiple Channels Design	197.8	181.4
Application-layer Dispatching Design	23.8	24.1

that the bottleneck of the distributed system is the computational part of the VD module, since the performance difference between the Campus and Home scenarios is not statistically significant.

The next section focuses on the Application-layer Dispatching Design only, since it produced the best performance by a long haul. The section verifies the maximum rate of full smart grid workflows that can be supported by the smart grid that was deployed.

5.3 Performance Under Steady State

The smart grid was put into a steady state by tweaking the period of the sensor messages that start up the full smart grid workflow. Starting from the data collected in the previous section, and with a few experiments aimed at tuning up the system, the smart grid was set up to process 40 and 130 full smart grid workflows per second when initiated by big sensor and small messages respectively.

The values of 130 and 40 are slightly lower than the ones suggested by the previous section, and for example the smart grid should have been able to cope with 150 full small grid workflows per second with small sensed data messages. Anyway, working that close to the maximum performance of the system led to high variances in the end-to-end delay, and the mean end-to-end delay grows to 5310 milliseconds (rate of 150) and to 1070 milliseconds (rate of 140), while for rates less or equal to 130 full smart grid workflows per second, the mean end-to-end delay keeps consistent with 310 milliseconds. As a title of example, Figure 7 and Figure 8 show the CDF (Cumulative Distribution functions) of the end-to-end delays in one experiment each. The experiments involved a total of 20000 full smart grid workflows initiated by short messages, with a rate of 140 and 130 full smart grid workflows per second, respectively.

The rest of the section considers 40 messages per second frequency for big messages and 130 messages per second for small messages. Table 3 reports the mean end-to-end delay experienced by the system over a full smart grid workflow. Even though the bandwidth for full smart grid workflows is the same, the mean delay for the home MPG is slightly larger since the ADSL line is slower than the fiber connection of the campus.

5.4 Discussion

From the data at hand, it is possible to dimension a system to cope with a given number of user premises. Current case studies suggest that a smart grid should

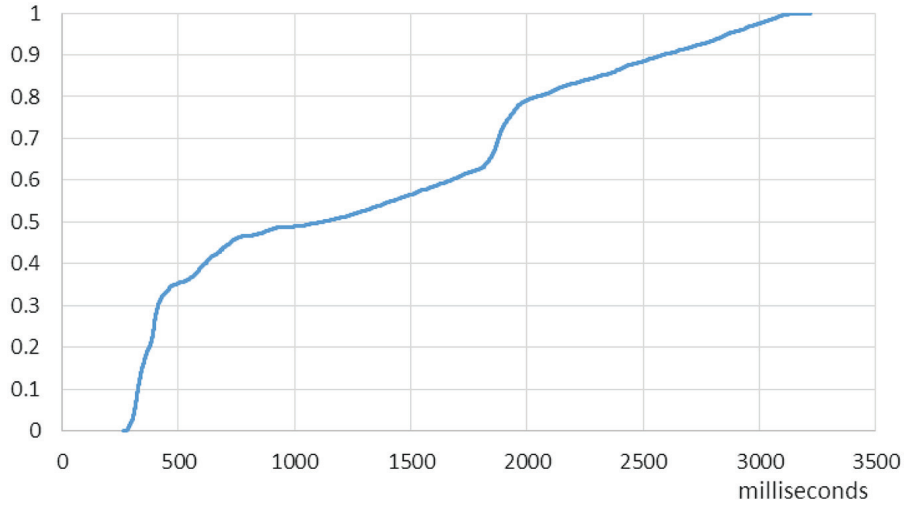

Figure 7 CDF of end-to-end delay, 140 small messages per second.

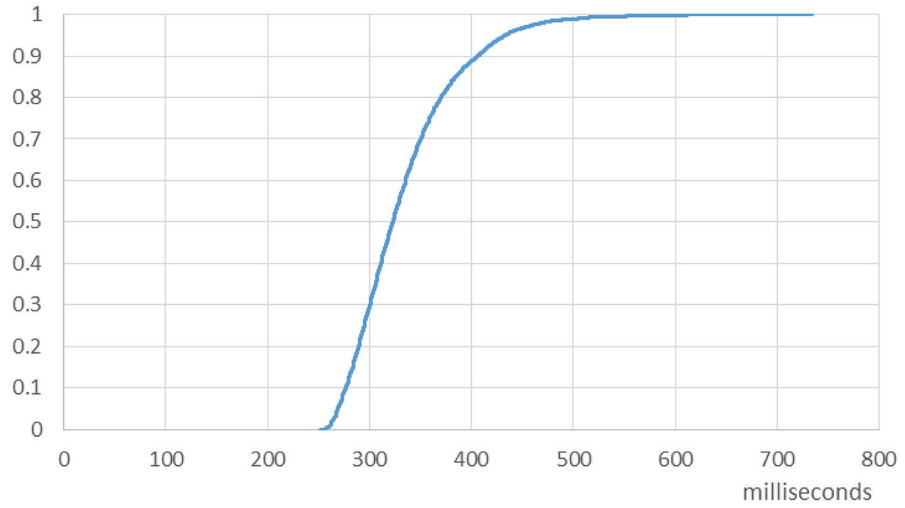

Figure 8 CDF of end-to-end delay, 130 small messages per second.

exchange one message with the user premises each 15, 30 or 60 minutes for monitoring only, and one message each 12, 8 or 4 seconds when Supervisory Control is taken into account [29, 30]. Let us consider the most aggressive option (one message each 4 seconds), considering full smart grid workflow initiated by small messages, each virtual machine running a VD module can

Table 3 Mean end-to-end delay, in milliseconds

	Campus [ms]	Home ADSL [ms]
Big messages	407	453
Small messages	310	331

take care of 520 HANs. Let us recall that the configuration of the virtual machine used in the experiments of this section is the same of the lower-end (t2.medium) virtual machines rented by Amazon [28], whose cost amounts to 0.060 $/hour. A VD installation to manage more than 500 houses would have a cost of less than 40 Euros/month. Considering a similar cost for the machine running the DB and the DB handler module, and for the machine running the SC module, we can cap the OPEX (Operating Expenses) of the smart grid under analysis to 120 Euros per month, which is less than 0.25 Euro per house. This analysis disregards other costs, for example it considers that each HAN has got internet access rented by the user for his own communication needs and thus the internet access does not contribute to the costs.

6 Conclusions

The smart grid presented in this work, and deployed in a number of demonstrators [6], proved to have a reasonable performance for the task at hand. A number of lessons have been taken by the work that led to its development:

- It is of foremost importance to use proper software architectures to enhance the computational performance of the smart grid.
- The documentation of some subsystems proposes either too conservative solutions (e.g.: creating a communication channel for each outbound message) or too liberal ones (e.g.: creating a thread for each device represented in the VD module, leading to excessive resource utilization).
- On the other hand, mainstream open source software (e.g.: the RabbitMQ messaging bus [24] and the JAXB library [23] based on DOM [25]) can suffice to provide the performance needed.
- Performance is strongly impacted by network usage. For example, Table 1 and Table 2 show that the “Application-layer Dispatching Design” is much faster than the previous design, and the difference between them is an optimization on how the modules access the messaging bus.
- Even though the thread structure of the systems has got a lower effect on the performance, it has profound impact on system stability, as it was testified by the “Massively Multi-threaded Design”.

- A stress test on the system can help for dimensioning the smart grid, since it provides a hint regarding the maximum message bandwidth of the system.
- If we consider a modern house that has got an ADSL connection, the cost for a service company to monitor the appliances in the HAN amounts to less than 0.25 Euro per month.
- Current technology is mature enough for the creation of companies focused on energy management for third parties, as proposed for the Service Provider domain of the smart grid in most meta-architecture models [1].

Future work will consider to compare the energy and money consumed for the smart grid to work, and the energy and money it can save. Planned work will also compare the ENCOURAGE smart grid with existing reference architectures and deployed systems.

Acknowledgment

This work was supported by EU Artemis JU funding, within the ENCOURAGE project, JU grant nr. 269354, and Arrowhead project, JU grant nr. 332987, and by National Funds in Portugal (through the Portuguese Foundation for Science and Technology) and Spain in the Artemis program.

References

[1] M. Albano, L. L. Ferreira, L. M. Pinho, "Convergence of Smart Grid ICT architectures for the last mile", IEEE Transactions on Industrial Informatics (TII), vol. 11, n. 1, pp. 187–197, February 2015

[2] M. Albano, L. L. Ferreira, L. M. Pinho, A. R. Alkhawaja, "Message-oriented middleware for smart grids", Computer Standards & Interfaces, Elsevier, vol.38, pp. 133–143 (2015)

[3] S. Appel, K. Sachs, A. Buchmann. "Towards benchmarking of AMQP." Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems. ACM, 2010

[4] J. L. Fernandes, I.C. Lopes, J. J. Rodrigues, S. Ullah. "Performance evaluation of RESTful web services and AMQP protocol." In Ubiquitous and Future Networks (ICUFN), 2013 Fifth International Conference on, pp. 810–815. IEEE, 2013.

[5] Aihkisalo, Tommi, and Tuomas Paaso. "A performance comparison of web service object marshalling and unmarshalling solutions." Services (SERVICES), 2011 IEEE World Congress on. IEEE, 2011

[6] M. Albano, L. L. Ferreira, T. Le Guilly, M. Ramiro, E. Faria, L. P. Dueñas, R. Ferreira, E. Gaylard, F. Pelegrin, E. Roarke, D. Lux, S. Scalari, S. M. Sørensen, M. Gangolells, L. M. Pinho, A. Skou, "The ENCOURAGE ICT architecture for heterogeneous smart grids", IEEE Eurocon Conference (Eurocon 2013), 1–4 July, 2013, Zagreb, Croatia

[7] RabbitMQ consortium, "RabbitMQ tutorial", available online at http://www.rabbitmq.com/getstarted.html, last accessed on 23/12/2015

[8] C. Teixeira, M. Albano, A. Skou, L. P. Dueñas, F. Antonacci, R. Ferreira, K. L. Pedersen, S. Scalari, "Convergence to the European Energy Policy in European countries: case studies and comparison," Social Technologies Research Journal, vol. 4, nº 1, pp. 7–18, 2014.

[9] Kabalci, Yasin. "A survey on smart metering and smart grid communication" Renewable and Sustainable Energy Reviews, 57: 302–318 (2016).

[10] Clastres, Cédric. "Smart grids: Another step towards competition, energy security and climate change objectives" Energy Policy, 39: 5399–5408 (2011).

[11] Malik, Farhan H. and Lehtonen, Mattí. "A review: Agents in smart grids" Electric Power Systems Research, 131: 71–79 (2016).

[12] Menniti, D., Sorrentino, N., Pinnarelli, A., Belli, G., Burgio, A., Vizza, P., "Local electricity market involving end-user distributed storage system", IEEE 15th International Conference on Environment and Electrical Engineering (EEEIC), pp. 384–388. (2015)

[13] Depuru, S. S. S. R., Wang, L., Devabhaktuni, V. "Smart meters for power grid: Challenges, issues, advantages and status" Renewable and Sustainable Energy Reviews, 15: 2736–2742 (2011).

[14] Zoha, Ahmed, et al. "Non-intrusive load monitoring approaches for disaggregated energy sensing: A survey." Sensors 12.12: 16838–16866 (2012).

[15] Colak, Ilhami, et al. "Smart grid projects in Europe: Current status, maturity and future scenarios." Applied Energy 152: 58–70 (2015).

[16] Carillo-Aparicio, Susana, Juan R. Heredia-Larrubia, and Francisco Perez-Hidalgo. "SmartCity Málaga, a real-living lab and its adaptation to electric vehicles in cities." Energy Policy 62: 774–779 (2013).

[17] G. Colouris, J. Dollimore e G. Blair, "Distributed Systems – Concept and Design 5th Edition", Morgan Kaufmann Publishers, pp. 1–33, 2012.

[18] M. K. Duncan e C., "AMQP 0.9.1 Model Explained," [Online]. Available: http://www.rubydoc.info/github/ruby-amqp/amqp/master/file/docs/AMQP091ModelExplained.textile
[19] AMQP Advanced Message Queuing Protocol, Protocol Specification, Version 0–9-1, 13 November 2008, http://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf, last accessed on 23/12/2015
[20] Barker, Sean, Aditya Mishra, David Irwin, Prashant Shenoy, and Jeannie Albrecht. "Smartcap: Flattening peak electricity demand in smart homes." In Pervasive Computing and Communications (PerCom), 2012 IEEE International Conference on, pp. 67–75. IEEE, 2012
[21] Chen, Hsinchun, Roger HL Chiang, and Veda C. Storey. "Business Intelligence and Analytics: From Big Data to Big Impact." MIS quarterly 36, no. 4 (2012): 1165–1188.
[22] International Electrotechnical Commission, "IEC 61968-11 – Common Information Model (CIM) Extensions for Distribution, ed. 2.0", 6 March 2013
[23] Fialli, Joseph, and Sekhar Vajjhala. "The Java architecture for XML binding (JAXB)." JSR Specification, January (2003)
[24] A. Videla, J. W. Williams, "RabbitMQ in Action: Distributed Messaging for Everyone" MEAP Edition, Manning Early Access Program, 2011
[25] Widener, P., Eisenhauer, G., Schwan, K., & Bustamante, F. E. Open metadata formats: efficient XML-based communication for high performance computing. Cluster Computing, 5(3), 315–324 (2002)
[26] Gamma, E., Helm, R., Johnson, R. and Vlissides, J., 1994. Design patterns: elements of reusable object-oriented software. Pearson Education.
[27] Broeer, T., Fuller, J., Tuffner, F., Chassin, D. and Djilali, N., 2014. Modeling framework and validation of a smart grid and demand response system for wind power integration. Applied Energy, 113, pp.199–207.
[28] http://aws.amazon.com/pt/ec2/pricing/
[29] Spanish Ministry of the Industry, Energy and Tourism, State Secretary for the Energy, Law IET/2013/2013, October 29^{th}, 2014, available online at: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2014–11274.
[30] Lawrence Berkeley National Laboratory, 2016. Future Opportunities and Challenges with Using Demand Response as a Resource in Distribution System Operation and Planning Activities. Document LBNL-1003951, available online at: https://emp.lbl.gov/sites/all/files/lbnl-1003951.pdf

Biographies

M. Macarulla is an Assistant Lecturer in the Department of Project and Construction Engineering at the Universitat Politècnica de Catalunya. He has completed his Ph.D degree from Universitat Politècnica de Catalunya in 2013 in the area of quality management.

He is currently working on the field of energy efficiency, focusing his research in the analysis of the building data coming from Energy Management Systems, and the development of Key Performance Indicators and their visualization. He was involved in 3 projects funded by the European Union and 2 project funded by the Spanish government. As a result of his research work, Dr. Marcel Macarulla has co-authored/edited 17 internationally refereed research papers.

M. Albano is Research Scientist in the CISTER Research Unit of the Polytechnic of Porto, Portugal, working on communication middleware for embedded systems with a focus on the application areas of smart grids, vehicular networks, and green wireless communications. He is a Founding Member of the Technical Committee on Green Communications and Computing (TCGCC).

Michele received his degree from the University of Pisa, Italy, has been involved in more than 10 European research projects, and he acted as technical manager for CELTIC project Green-T and work package leader for FP7 IP ROMEO and ITEA2 CarCoDe.

His works have been published in more than 70 international conferences and journals, he is on the editorial board of the International Journal of Social Technologies and of the Transactions on Emerging Telecommunication Technologies since 2011, and in 2015 he has been appointed as Editor in Chief for the Journal of Green Engineering.

L. L. Ferreira has a MSc (1997) and a Ph.D. (2005) in Electrical and Computer Engineering at the University of Porto. Since 1996 he is a professor at the Department of Computer Engineering, School of Engineering of the Polytechnic Institute of Porto. He is a Research Associate at the CISTER (Research Centre in Real-Time and Embedded Computing Systems) research unit, where he currently leads Arrowhead and MANTIS European projects. He has been working in the field of distributed embedded real-time systems for more than 20 years, having participated in more than 15 projects (national and international). He has also participated on several conferences as program co-chair and published more than 50 papers in international conferences and journals.

C. Teixeira, born in 1987, holds a degree (2010) in Computer Science Engineering from the School of Engineering, Polytechnic Institute of Oporto. He was involved in the European projects ENCOURAGE and Arrowhead. Whilst working full time on the aforementioned ENCOURAGE Project and

Arrowhead Project in CISTER Research Unit, he completed his Masters (2015) in Computer Science Engineering at School of Engineering, Polytechnic of Oporto, in the area of Architectures, Systems and Networks. His main interests are in the field of Networking, Mobile development, Multi-Agent Systems and Web development.